6.1.4. Portable C Code

The source codes generated by the *fuzzy*TECH C code generator are portable. Thus, fuzzy runtime systems can be implemented on any target platform for which a C compiler exists. Any code generated by *fuzzy*TECH can be used royalty-free. This applies also to the library, as long as it has not been modified and is part of a project that was generated by *fuzzy*TECH.

Overview

An implementation of a fuzzy runtime system consists of at least three modules:

1. A fuzzy library, that stores the basic fuzzy algorithms which are independent from any fuzzy system.
2. One or more fuzzy system modules generated by the C code generator of *fuzzy*TECH. Each module contains code and data for its specific fuzzy logic system.
3. One or more of your own software modules for implementing the application, that uses the fuzzy logic system.
4. A communication module, that implements one of the communication channels supported by *fuzzy*TECH. Note that this applies only to the *fuzzy*TECH Online Edition. Even for this edition the usage of a communication module is optional.

Figures 162 and 163 illustrate the hierarchy of these modules:

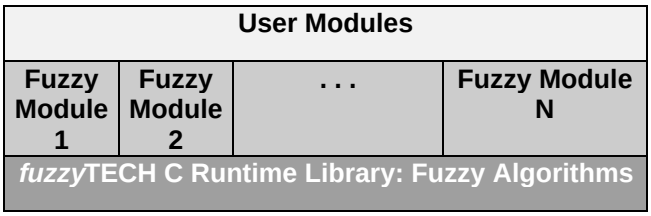

Figure 162: Fuzzy Runtime Systems in C Language

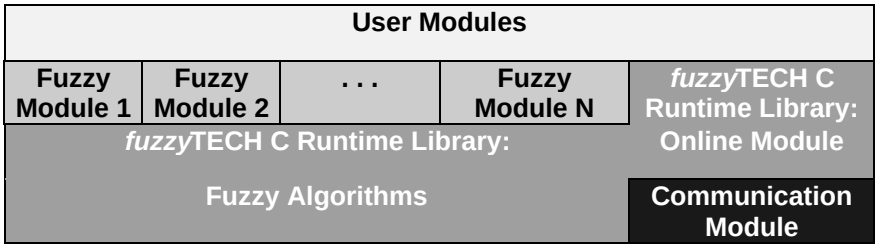

Figure 163: Fuzzy Runtime Systems And Online in C Language

The steps to get an application to run and compute one or more fuzzy logic systems are described in this section as follows: Section 6.1.4.1 explains the steps to build your own fuzzy library. Section 6.1.4.2 describes the integration of the user interface of the fuzzy system modules into your own source codes. Users of the *fuzzy*TECH Online Edition may need to integrate a communication module to support the online connection between the fuzzy runtime system and the *fuzzy*TECH Shell. Refer to Section 6.1.4.3 for more details. Section 6.1.4.4 describes the process of linking all modules together to get an application running.

6.1.4.1. Building the *fuzzy*TECH C Runtime Library

Build Your Own Library

The library used by the C code generator is delivered as source code to keep the C Code based fuzzy runtime systems portable to any platform. Therefore you have to create the binary form of the library yourself. Building your own *fuzzy*TECH C Runtime Library allows you to use *your* favorite compiler with *your* favorite compiler settings for *your* favorite target CPU.

Note: You can use any C compiler for any target CPU. Neither does the *fuzzy*TECH C Runtime Library need other libraries nor does it include files of your compiler.
Note: The *fuzzy*TECH C Runtime Library has to be built for your personal environment only once. Afterwards, it can be used for all fuzzy runtime systems on this environment.

Directory

You should locate your library in the directory …`\RUNTIME/C/LIB`. All necessary source codes for building the library can be found in the directory …`\RUNTIME/C/LIB/SRC`.

To do

Due to the great number of different C compilers, this section is a rather general description and not a step-by-step guidance. It may be necessary to browse your compiler manual to realize the building steps. The library consists of many modules. You will find one C source file for each module.

1. Important: Set the fuzzy library preprocessor definitions. Please refer to Section 6.1.4.1.1 for details.
2. Compile all `*.C` source files to `*.OBJ` binary object files.
3. Move all `*.OBJ` files to the library.

Please refer to the manual of your compiler, if you are not firm with these steps.
Note: You can use both the command line version and, if available, the integrated development environment (IDE) of your compiler. For some compiler tool kits the library manager may be a separate tool.

Compiler Settings

Due to the great number of very different C compilers, this section can give only some general hints. Some compilers offer options for optimizing the library

either for size or for speed. Note: Some compilers may support different memory models (small, medium, large, ...). Make sure that the fuzzy library was compiled for the same memory model as your application to be linked with the fuzzy library. If you want to use different memory models you have to build separate libraries for each memory model. Please refer to your compiler manual for more details.

Sample

The batch file `FTMAKE.BAT` creates the fuzzy library for the PC based C compilers of Microsoft or Borland. It is based on the command line versions of both compilers. You may use this file as sample for adapting your compiler.

6.1.4.1.1. Preprocessor Definitions

What Is It?

Preprocessor definitions are standard control elements in the C language to build if-else-constructs that are evaluated at compile time. The same source code can thus be used for multiple purposes. C programmers generally use preprocessor definitions together with the statements: `#define` and `#ifdef`.

Why Is Their Usage Essential?

Different *fuzzy*TECH Editions need different *fuzzy*TECH C Runtime Libraries. The building process for the different libraries is controlled by preprocessor definitions. You have to set one or more preprocessor definitions to build a library that matches the C code generated by your *fuzzy*TECH Edition. Some Editions may generate C code for different computation resolutions. The computation resolution is determined by the base variable type selected in the Project Options dialog Global. These editions need two libraries, one for 8-bit resolution and one for 16-bit or 'double' resolution.

Where Can They Be Set?

Preprocessor definitions on compile time can either be set in the command line of your compiler or in a dialog of your integrated development environment. Please refer to the manual of your compiler.
Note: Most C compilers support in the command line version the switch `-D` or `/D` to set a preprocessor definition.

Which Shall Be Set?

Refer to the tables below to find out which preprocessor definition must be set for which library and which edition.

Edition	**Library Name**	**Preprocessor Definitions**	**(optional)**
Professional	FTC16.LIB FTC8.LIB	FTLIBC16 PRECOMPILER FTLIBC8 PRECOMPILER	FT_KRC FT_KRC
Online	FTC16.LIB FTC8.LIB	FTLIBC16 ONLINE FTLIBC8 ONLINE	FT_KRC FT_KRC
MCU-C, MCU-MP	FTC16.LIB FTC8.LIB	FTLIBC16 FTLIBC8	
MCU-166, MCU-320, MCU-96	FTC16.LIB	FTLIBC16	
MCU-51, MCU-ST6, MCU-374, MCU-11/12	FTC8.LIB	FTLIBC8	

Preprocessor Definition	Description
FTLIBC16	Enables 16 bit computation resolution. Note: The double interface of the code generator uses 16 bit resolution too.
FTLIBC8	Enables 8 bit computation resolution.
PRECOMPILER	Enables additional data used by the non MCU fuzzy algorithms.
FT_KRC	Enables the support of Kernighan&Ritchie C compilers. The default is ANSI-C.

Sample

Assuming you use the *fuzzy*TECH Professional Edition and the Microsoft Visual C/C++ Compiler. In order to create a library with 16 bit computation resolution you may compile all C files in the directory `RUNTIME/C/LIB/SRC` with the following command line call:

```
cl.exe -c -DFTLIBC16 -DPRECOMPILER *.C
```

where `cl.exe` is the call to the Microsoft compiler, `-c` is the compiler command for generating object files only, `-D` is the command to set the preprocessor definitions and `*.C` is the wildcard for all C source files in the current directory.

6.1.4.2. Using the Generated Module

Generate the C Module

This section shows how C code generated by *fuzzy*TECH can be integrated in your own software modules. Select "Tools\Compile to...\C..." in the main menu of *fuzzy*TECH to generate the current fuzzy logic project as C code. The code generator creates two files with the same name as the fuzzy system, one with the file extension `*.C`, the other one with the extension `*.H`. The `*.C` file contains code and data of the fuzzy logic system, the `*.H` file contains the export interface of this module. Note, that the name of each exported function or variable in this `*.H` file ends with the name of the fuzzy logic system. Note that the generated modules use the project independent fuzzy logic algorithms of the fuzzy library. Assuming you want to integrate two different fuzzy logic systems in your application. Furthermore the fuzzy logic systems have the names MYPROJ1.FTL and MYPROJ2.FTL. Generating the C-Modules for both systems will create the files:

```
MYPROJ1.C, MYPROJ1.H
MYPROJ2.C, MYPROJ2.H
```

Overview

The following enumeration gives an overview about the steps necessary to integrate a fuzzy system module into the source code of your software module(s):

1. Include C header files
   - Include the export interface(s) of the fuzzy system module(s).
   - *fuzzy*TECH Online Edition only: Include the export interface of the fuzzy online module.
2. Initialization during startup of your application
   - *fuzzy*TECH Online Edition only: Initialize the fuzzy online module.
   - Initialize the fuzzy system module(s).
3. Computation of each fuzzy system module:
   - Set the input variables.
   - Call the fuzzy computation.
   - Get the output variables.
4. *fuzzy*TECH Online Edition only: Call the fuzzy online communication.
5. *fuzzy*TECH Online Edition only: Trace control allows you to start or stop the trace process with your own logical expressions.

Include Header Files

You have to include a `*.H` file for each fuzzy system module in your C source code. Users of the *fuzzy*TECH Online Edition have additionally to include the file `ONLINE.H` which is located at `RUNTIME\C\INCLUDE`. Add the following lines to the include section of your source code.

```
/*all fuzzyTECH Editions                          */
/*include the export interfaces of the fuzzy systems*/
#include "myproj1.h"               /*fuzzy system 1*/
#include "myproj2.h"               /*fuzzy system 2*/

/*fuzzyTECH Online Edition only                   */
/*include the export interface of the online module */
#ifdef FT_ONLINE
#include "online.h"          /*fuzzy online manager*/
#endif
```

Initialization

Each fuzzy system module must be initialized only once during the startup of the application. The initialization function has no parameters and no return value. Users of the *fuzzy*TECH Online Edition have to initialize the fuzzy online module additionally. It's important to do this before initializing the fuzzy system modules. Add the following lines to the part of your source that contains the startup sequence.

```
/*fuzzyTECH Online Edition only                   */
/*initialize the fuzzy online manager             */
#ifdef FT_ONLINE
initonline();                /* must be done first */
#endif

/*all fuzzyTECH Editions                          */
/*initialize the fuzzy system modules             */
initmyproj1();
initmyproj2();
```

Fuzzy Computation: Parameter Passing

The *fuzzy*TECH C code generator supports two different possibilities of passing the input and output variable values between your source code and the fuzzy system module. This option is controlled by the Public Input and Output checkbox in the dialog "Project Options/Code Generator".
Depending on your setting you get C code with a computation function using:

1. Function Parameters
2. Global Variables.

Note that global variables will probably be easier to handle, if you change the number of input or output variables of your fuzzy logic system. The resulting code may also be a little bit faster.

Fuzzy Computation: Data Type and Range of the Variables

The *fuzzy*TECH C code generator supports different data types for input and output variables of the fuzzy system. The data type is defined by the radio buttons Base Variable Data Type in the dialog "Project Options/Global". Depending on the setting in this dialog you get C code using variables with

1. Data type `FUZZY`, which is 8 bit unsigned integer.
   Value range: The **code values** set in the Variables Properties: Base dialog.
2. Data type `FUZZY`, which is 16 bit unsigned integer.
   Value range: The **code values** set in the Variables Properties: Base dialog.
3. Data type `double`, which is the C standard type.
   Value range: The **shell values** set in the Variables Properties: Base dialog.

Fuzzy Computation: Function Prototype

The function name for the fuzzy computation is in any case the same as given for the fuzzy logic system itself. Depending on the chosen parameter passing and data type options the fuzzy computation function exported in the `*.H` file will have different prototypes.

1. Global variables, all data types:
```
FLAGS myproj1(void);
```
2. Function parameters, data type FUZZY:
```
FLAGS myproj1(FUZZY in1, FUZZY in2, ...,
              FUZZY* out1, FUZZY* out2, ...);
```
3. Function parameters, data type double:
```
FLAGS myproj1(double in1, double in2, ...,
              double* out1, double* out2, ...);
```

If you use global variables the `*.H` file exports all input and output variables of the fuzzy logic system with the correct data type automatically. The names of these variables are constant. They are based on the names you have used in *fuzzy*TECH and follow this scheme:

```
<Variable Name>_<Module Name>
<Term Name>_<Variable Name>_<Module Name>
```

If you use the function parameter interface, you have to declare variables in your source code. One variable for each input and each output of the fuzzy logic is recommended. You are free to use any names for these variables, but their data type has to be either `double` or `FUZZY`. When calling the computation function, you have to place your variables at the correct position that matches the fuzzy logic system. In any case the function needs first the inputs. They are in alphanumeric order sorted by the variable names used in *fuzzy*TECH. The function expects call-by-reference parameters after the inputs. This is one pointer for each output variable in alphanumeric order.

Fuzzy Computation: Function Call

Assuming that the code of MYPROJ1 uses function parameters, the code of MYPROJ2 uses global variables, and both fuzzy systems use the data type `FUZZY`, your source code may look like the following pseudo C code:

```
/*i/o-handling and computation                        */
/*fuzzy system MYPROJ1 with function parameters      */
FLAGS flags;
/*declare some variables of data type FUZZY
FUZZY a, b, c, ...; /* your variables */
FUZZY x, y, z, ...; /* your variables */
...
/*compute the fuzzy system */
flags = myproj1(a, b, c, ...,&x, &y, &z, ...);
...
...
...
/*i/o-handling and computation                        */
/*fuzzy system MYPROJ2 with global variables         */
/*set all input variables of the fuzzy system */
in1_myproj2 = ...; /* variables exported by MYPROJ.H */
in2_myproj2 = ...; /* variables exported by MYPROJ.H */
in3_myproj2 = ...; /* variables exported by MYPROJ.H */
...
/*compute the fuzzy system */
flags = myproj2();
/*use the output variables of the fuzzy system */
... = out1_myproj2; /* variables exported by MYPROJ.H */
... = out2_myproj2; /* variables exported by MYPROJ.H */
... = out3_myproj2; /* variables exported by MYPROJ.H */
...
```

Fuzzy Computation: Return Value

The fuzzy computation function returns a value of the type `FLAGS` with the fuzzy logic inference control flags. Each bit in `FLAGS` represents one output variable, starting with the lowest order bit and assigned in sequence to the output variables in alphanumeric order. A zero bit for a variable indicates that for this output variable at least one rule has fired. Hence, a return value of zero indicates that for every output variable at least one rule has fired. The number of bits of the data type `FLAGS` depends on the *fuzzy*TECH Edition:

*fuzzy*TECH Edition	FLAGS
Professional, Online	32 bit integer
MCU-96, MCU-166, MCU-320	16 bit integer
MCU-C, MCU-ST6, MCU-MP, MCU-51, MCU-11/12	8 bit integer

Consider a fuzzy logic system with four (4) output variables: A_Output, B_Output, C_Output and D_Output. For this system, the bits 3...0 of the return variable correspond to variable A_Output ... D_Output. All higher order bits are not used. A return value of 5, evaluated in a specific control cycle, indicates that no rule has fired for the second and the forth output variable.

32-bit value	FLAGS flags=5	Variables in Output Interface	Output Value
MSB: 31	0	not used	-
30	0	not used	-
29	0	not used	-
28	0	not used	-
27	0	not used	-
	...	...	...
...	0	not used	-
	...	...	...
4	0	not used	-
3	0	A_Output	calculated
2	1	B_Output	default-value
1	0	C_Output	calculated
LSB: 0	1	D_Output	default-value

Figure 164: Return Flags of a Return Value 5 for a System With Four Output Variables (32-bit FLAGS type)

Online Communication

The *fuzzy*TECH Online Edition has the option to enable an online connection with your fuzzy runtime system. Add a call to the fuzzy online module in your source code to support this communication mechanism.

```
/*fuzzyTECH Online Edition only                    */
#ifdef FT_ONLINE
online();              /* call the online module   */
#endif
```

Note that you must call this function regularly and as often as possible. Otherwise the online communication will run into a time-out. For example, if your application has a kind of main control loop, this would be the best place to insert the function call.

Note that the function `online()` should not interrupt the computation of the fuzzy systems. This is important, in case you use a timer interrupt to call this function.

Note that the function `online()` is programmed in such a way that it consumes as little time as possible. During an active online connection it copies the maximum number of 76 bytes and sends an answer to *fuzzy*TECH before it returns. If no active online connection is running, the function returns immediately.

Trace Control

The *fuzzy*TECH Online Edition offers the option to generate C code supporting a trace buffer that records the input and output values of the fuzzy system in real-time in your application. During an online connection the trace process is controlled by the Trace Control dialog. The dialog offers an option to start or stop the trace process by external trigger events. Therefore the generated C code exports two additional functions that control the trace process. Call these functions in your source code to start or stop the trace at arbitrary logical conditions. Note that once the Trace process has been started, multiple calls of the following start and stopp function are ignored:

```
/*fuzzyTECH Online Edition only                    */
/*evaluate your own logical conditions             */
/*to start or stop the trace process               */
#ifdef FT_ONLINE
if((fCond1 && fCond2) || fCond3) /*this is a sample!*/
  StartTracemyproj1();
else
  StopTracemyproj1();
#endif
```

Code Samples

The following pseudo C code samples illustrates the user interface of the fuzzy modules for different settings of the code generator options, i.e. base variable data type and parameter passing.

```
                    /* Example 1: MYMAIN.C */
  /* single fuzzy project, function parameters, integer data type */

#include "myproj.h"                    /* include fuzzy system     */
...                                    /* more includes            */
...                                    /* more declarations        */
void main(void) {                      /* main program             */
  FUZZY myin1, myin2,..;               /* declare local variables  */
  FUZZY myout1, myout2,..;             /* declare local variables  */
  FLAGS rv;                            /* declare return value     */
  ...                                  /* more code                */
  initmyproj();                        /* initialize fuzzy system  */
  ...                                  /* more code                */
  while(!stop) {                       /* control loop             */
    ...                                /* more code                */
    myin1 = ...                        /* set the input variable   */
    myin2 = ...                        /* set the input variable   */
    ...                                /* more code                */
    rv = myproj(myin1, myin2,..,       /* call the fuzzy system    */
                &myout1, &myout2,..);  /*                          */
    ... = myout1;                      /* use the output variables */
    ... = myout2;                      /* use the output variables */
    ...                                /* more code                */
  }                                    /* end of control loop      */
...                                    /* more code                */
}                                      /* end of main program      */
```

```
                    /* Example 2: MYMAIN.C */
   /* single fuzzy project, function parameters, double data type */

#include "myproj.h"                  /* include fuzzy system     */
...                                  /* more includes            */
...                                  /* more declarations        */
void main(void) {                    /* main program             */
  double myin1, myin2,..;            /* declare local variables  */
  double myout1, myout2,..;          /* declare local variables  */
  FLAGS rv;                          /* declare return value     */
  ...                                /* more code                */
  initmyproj();                      /* initialize fuzzy system  */
  ...                                /* more code                */
  while(!stop) {                     /* control loop             */
    ...                              /* more code                */
    myin1 = ...                      /* set the input variable   */
    myin2 = ...                      /* set the input variable   */
    ...                              /* more code                */
    rv = myproj(myin1, myin2,..,     /* call the fuzzy system    */
                &myout1, &myout2,..); /*                         */
    ... = myout1;                    /* use the output variables */
    ... = myout2;                    /* use the output variables */
    ...                              /* more code                */
  }                                  /* end of control loop      */
...                                  /* more code                */
}                                    /* end of main program      */
```

```
                          /* Example 3: MYMAIN.C */
                /* multiple fuzzy systems, global variables, */
                /* any data type, online support             */
#include "myproj1.h"                        /* include fuzzy system 1   */
 #include "myproj2.h"                       /* include fuzzy system 2   */
 ...                                        /* more fuzzy systems       */
 #ifdef FT_ONLINE                           /*                          */
 #include "online.h"                        /* include online module    */
 #endif                                     /*                          */
...                                         /* more includes etc.       */
 void main(void) {                          /* main program             */
   FLAGS rv;                                /* declare return value     */
  ...                                       /* more code                */
   #ifdef FT_ONLINE                         /*                          */
   initonline();                            /* initialize online module */
   #endif                                   /*                          */
   initmyproj1();                           /* initialize fuzzy system 1*/
   initmyproj2();                           /* initialize fuzzy system 2*/
   ...                                      /* initialize other systems */
   ...                                      /* more code                */
   while(TRUE) {                            /* control loop             */
     ...                                    /* more code                */
     myin1_myproj1 = ...;                   /* set input data ...       */
     myin2_myproj1 = ...;                   /*...for fuzzy system 1     */
     ...                                    /*...more inputs            */
     rv=myproj1();                          /* call fuzzy system 1      */
     ... = myout1_myproj1;                  /* transfer output data ... */
     ... = myout2_myproj1;                  /*...to process             */
     ...                                    /*...more outputs           */
     ...                                    /* more code                */
     myin1_myproj2 = ...;                   /* set input data ...       */
     myin2_myproj2 = ...;                   /*...for fuzzy system 2     */
     ...                                    /*...more inputs            */
     rv=myproj2();                          /* call fuzzy system 2      */
     ... = myout1_myproj2;                  /* transfer output data ... */
     ... = myout2_myproj2;                  /*...to process             */
     ...                                    /*...more outputs           */
     #ifdef FT_ONLINE                       /*                          */
     online();                              /* call online module       */
     #endif                                 /*                          */
     ...                                    /* more code                */
   }                                        /* end of control loop      */
}                                           /* end of main program      */
```

6.1.4.3. Online Communication Module

Online Communication Channels

The *fuzzy*TECH Online Edition offers the option to enable an online connection with your fuzzy runtime system. The *fuzzy*TECH Shell is running on a MS Windows operation system while your fuzzy runtime system may run on a different hardware platform. If you want to connect both systems, you have to decide, which communication channel is the best for your purpose. In every case both applications must use the same communication channel with the same settings.

Communication Channels of fuzzyTECH

*fuzzy*TECH supports different standard communication channels as the serial interface, shared file systems and IPX/SPX. Note, that *fuzzy*TECH offers an open interface. You may write your own DLL that uses other communication channels. Refer to Section 7.2 for details.

Communication Channels of Fuzzy Runtime System

The main modules of the fuzzy online manager are part of the fuzzy library. This does not include the layer that accesses to the hardware of the communication channel. It is separated due to portability reasons. The module consists of two files named `COMM.C` and `COMM.H` and has a fix interface that is used by the fuzzy library. You will find source codes for some communication modules in the sub-directories of `…\RUNTIME\C\LIB\SRC\ONLINE\COMM`. The file `README.TXT` in this directory contains additional information. You will find ready-to-go modules for:

1. MS DOS: Serial Interface
2. OS/2: Serial Interface
3. All: Shared File System

If you have decided which channel is the best for your target system, it may be necessary to change the original source code to set your preferred parameters. For example you may change the settings for the serial interface or you may change the communication directory for shared file systems.

User Defined Modules

If you use an other platform for your fuzzy runtime system, you have to implement your own communication module. A prototype can be found in `…\RUNTIME\C\LIB\SRC\ONLINE\COMM\USER\COMM.C`. It contains four empty functions that

1. initialize and open the communication channel,
2. close the communication channel,
3. transmit bytes to the communication channel,
4. receive bytes from the communication channel.

Please follow the instructions in the comments of the functions. Note, that you should not change the function prototypes. If you need help or more information, do not hesitate to contact our technical support.

6.1.4.4. Compiling and Linking

This chapter describes the compiling and linking process of all modules that are part of the fuzzy runtime system. Due to the great number of different C compilers, this section is a rather general description and not a step-by-step guidance. It may be necessary to browse your compiler manual to realize the building steps.

If you implement a new application please follow the steps of the enumeration below. If you add a fuzzy runtime system into an existing application please proceed analogously.

1. Check, whether your C compiler is installed correctly.
2. Check, whether you have built the fuzzy library.
3. Create a new project or makefile for your application.
   - Add your software module(s) to the project.
   - Add the fuzzy module(s) to the project.
     If your edition supports different data types, please note that it's not possible to integrate 16 bit or double fuzzy modules together with 8 bit fuzzy modules in the same application.
   - Add the correct fuzzy library to your project.
     `FTC16.LIB` for fuzzy modules with data type 16 bit or double.
     `FTC8.LIB` for fuzzy module with data type 8 bit.
   - Add the communication module to your project (Online Edition only).
   - Add the directory …`\RUNTIME\C\INCLUDE` to the include file search path of your compiler.
   - Add the directory …`\RUNTIME\C\LIB` to the library search path of your compiler.
4. Build the application.
   - Compile your own module(s).
   - Compile the fuzzy module(s).
   - Compile the communication module (Online Edition only).
   - Link all object files and libraries.
5. Execute the application.

Figure 165 illustrates the build process for the *fuzzy*TECH Online Edition. Note that all other editions do not need the communication module.

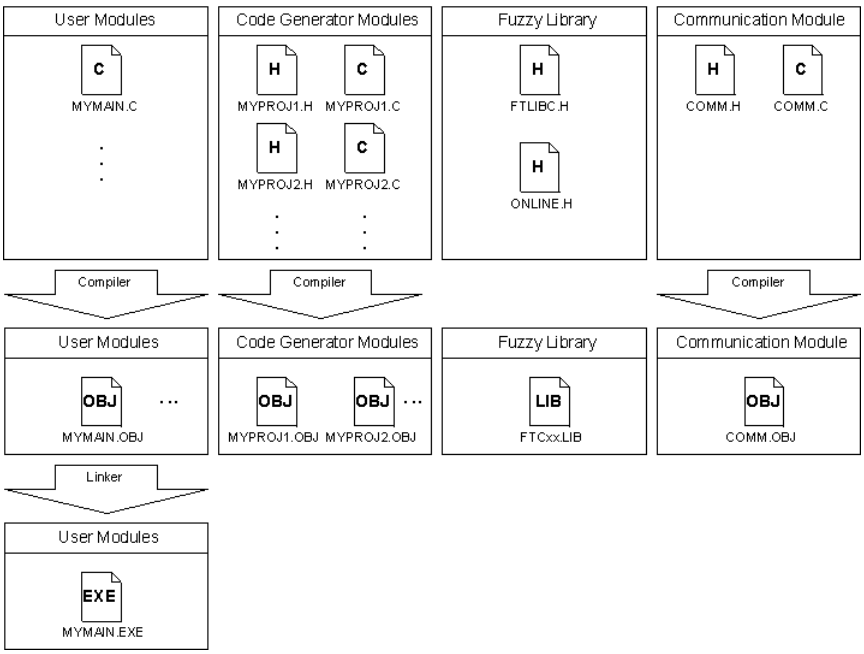

Figure 165: Building an Application with Fuzzy Runtime Systems and Online

6.1.4.5. Examples

All *fuzzy*TECH Editions generating C code (refer to Table 5 of Section) come with at least one sample that shows the integration of the generated code in user modules. The samples are located in the sub-directories of …`\RUNTIME\C\SAMPLES`. More details about the samples can be found in `README.TXT`.

MYMAIN Sample

This sample aims only to illustrate the user interface between your own code and the generated fuzzy module. Note, that most parts of the source code are dummies. There is no kind of display or interaction if you run the sample.

Directory and Files

This sample is located at …`\RUNTIME\C\SAMPLES\MYPROJ`.
It consists of following files:

`MYPROJ.FTL`	fuzzy system
`MYMAIN.C`	your application code
`MYPROJ.C`	fuzzy module
`MYPROJ.H`	fuzzy module export interface
`FTC16.LIB`	your fuzzy library

Build Procedure For compilers of Microsoft or Borland the batch file BUILD.BAT demonstrates all steps necessary to build an running application.